A Hybrid IP Lookup Architecture with Fast Updates

Layong Luo[1,2], Gaogang Xie[1,*], Yingke Xie[1], Laurent Mathy[3], Kavé Salamatian[4]
[1]Institute of Computing Technology, Chinese Academy of Sciences, China
[2]Graduate University of Chinese Academy of Sciences, China
[3]Lancaster University, United Kingdom. University of Liège, Belgium
[4]University of Savoie, France
{luolayong, xie, ykxie}@ict.ac.cn, laurent@comp.lancs.ac.uk, kave.salamatian@univ-savoie.fr
*Corresponding Author: xie@ict.ac.cn

***Abstract*—As network link rates are being pushed beyond 40 Gbps, IP lookup in high-speed routers is moving to hardware. The TCAM (Ternary Content Addressable Memory)-based IP lookup engine and the SRAM (Static Random Access Memory)-based IP lookup pipeline are the two most common ways to achieve high throughput. However, route updates in both engines degrade lookup performance and may lead to packet drops. Moreover, there is a growing interest in virtual IP routers where more frequent updates happen. Finding solutions that achieve both fast lookup and low update overhead becomes critical. In this paper, we propose a hybrid IP lookup architecture to address this challenge. The architecture is based on an efficient trie partitioning scheme that divides the Forwarding Information Base (FIB) into two prefix sets: a large disjoint leaf prefix set mapped into an external TCAM-based lookup engine and a small overlapping prefix set mapped into an on-chip SRAM-based lookup pipeline. Critical optimizations are developed on both IP lookup engines to reduce the update overhead. We show how to extend the proposed hybrid architecture to support virtual routers. Our implementation shows a throughput of 250 million lookups per second (MLPS). The update overhead is significantly lower than that of previous work and the utilization ratio of most external TCAMs is up to 100%.**

I. INTRODUCTION

IP lookup is a critical function of Internet routers. Since the introduction of CIDR (Classless Inter-Domain Routing) in 1993, finding the next hop for a destination IP address has become a longest prefix matching (LPM) problem. Indeed, given a destination address, multiple IP address prefixes of different lengths may exist, in the Forwarding Information Base (FIB) of the router, that match (*i.e.* contain) the given address and the longest such prefix must be used to determine the next hop for the corresponding packet to ensure correct forwarding operation.

The longest prefix matching problem lends itself to a hierarchical data structure for which a trie is an efficient representation (see Fig. 1(a)). In the context of IP lookup, a trie contains two types of nodes: 1) nodes (which we call prefix nodes and are shown as dark nodes in Fig. 1(a)) that represent predefined prefixes for which valid next hop information exists; and 2) nodes (which we call non-prefix nodes and are drawn clear) that do not contain next hop information. The longest prefix matching a destination address is then determined by following a single path from the trie root, with the longest-prefix match corresponding to the last prefix node encountered before the end of the path. Note that any encountered leaf node will contain a longest-prefix match. Moreover, the address space represented by the prefix stored at a node is always contained within the address space represented by the prefix stored at its ancestor nodes. Nonetheless, as there is only one leaf node per trie-path, prefixes stored at different leaf nodes are disjoint, *i.e.,* the corresponding address spaces of two leaves have no address in common.

As network link rates are being pushed beyond 40 Gbps, IP lookup with LPM becomes a major bottleneck in high-speed routers. The high lookup performance required by such high link rates is hard to be achieved in software [1] and two major hardware implementation techniques have been used to achieve such high performance: TCAM (Ternary Content Addressable Memory)-based lookup engines and SRAM (Static Random Access Memory)-based lookup pipelines.

A TCAM implements a high-speed associative memory, where in a single clock cycle a search key is compared simultaneously with all the entries (*i.e.*, keys) stored in the TCAM to determine a match and output the address of it. As TCAM entries can be specified using three states (0, 1, and 'X' meaning don't care), this type of memory is particularly well suited for storing IP prefixes where masked bits are given 'X' states. Indeed, because of the 'X' bits, several TCAM entries could match a given IP address, so TCAMs are designed to always return the first matching entry encountered (TCAM entries have an intrinsic order represented by an address). Therefore, in order to provide correct LPM operations, prefixes are stored in the TCAM with reverse order in overlap, *i.e.*, longest prefix should be stored first. These order constraints result in a large number of TCAM entry movements on some route updates, with large impact on the lookup performance and possible packet drops[2].

Because of the interest of the TCAM and the importance of the problems solved by it, several research efforts have led to new algorithms to solve the issue of TCAM updates. In [3], two approaches named PLO_OPT and CAO_OPT have been proposed. PLO_OPT maintains the prefix-length order by putting all the prefixes in order of decreasing prefix lengths and keeping the unused space in the center of a TCAM. CAO_OPT relaxes the constraint to only overlapping prefixes in the same chain (a single path from the trie root). Both of the algorithms can decrease the number of entry movements per update. However, multiple entry movements are still needed for one

978-1-4673-0775-8/12/$31.00 ©2012 IEEE

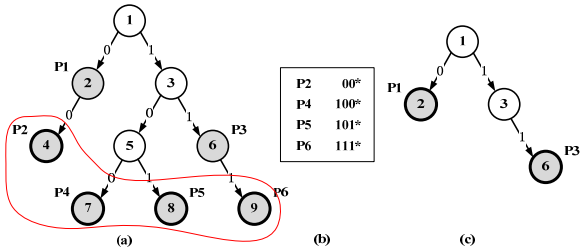


Figure 1. (a) A sample trie, (b) the corresponding disjoint prefix set, and (c) the corresponding overlapping prefix set (a small trie).

route update in the worst case[3]. In another approach, order constraints can be completely avoided in a TCAM, by converting the whole prefix set into an equivalent minimum independent prefix set (MIPS) [4] using the leaf pushing technique [5]. However, leaf pushing may lead to prefix duplication. When a prefix is updated, all of its duplicates should also be modified. Therefore, multiple write accesses may still be needed during a single route update. Additionally, TCAM updates can be performed without packet drops by duplicating the TCAM, with updates done to the shadow TCAM and the active one swapped out. However, the TCAM memory requirements are double.

The other major hardware implementation technique is the SRAM-based lookup pipeline[6], which corresponds to a straightforward mapping of each trie level onto a corresponding pipeline stage with its own SRAM memory, in order to achieve a throughput of one lookup per clock cycle through the pipeline. In such solutions, the number of pipeline stages depends on the stride used (*i.e.*, the number of bits used to determine which branch to take at each stage -- in Fig. 1 and in the rest of this paper we use 1-bit strides). Therefore, the lookup pipeline will require a rather high number of separate SRAMs (up to 32 in the case of IPv4). The Field Programmable Gate Array (FPGA) is a natural hardware choice for implementation of the SRAM-based pipeline, as it contains hundreds of separate SRAMs. Nevertheless, the on-chip SRAM is generally a scarce resource which should be allocated and utilized efficiently, or otherwise be complemented by external SRAMs[7]. One major issue here is that assigning the right size of the SRAM to each stage of the pipeline and utilizing each SRAM efficiently is complicated by the fact that it intimately depends on the shape of the trie. While much work has been devoted to this issue[8-10], the fact remains that on-chip SRAM is still unable to accommodate the typically large inter-domain FIB (as shown in TABLE I, about 360K prefixes to date). For example, in [10] it is reported that OLP(Optimized Linear Pipeline) can support 30K IPv4 prefixes using 3.456 Mb of on-chip SRAMs. Hence, given a state-of-the-art large Virtex-6 FPGA (*e.g.*, XC6VHX565T) with 32 Mb of on-chip SRAMs, only about 277K IPv4 prefixes can be stored using OLP. This means that the memory size is still a challenge in the SRAM-based lookup pipeline.

Route updates are handled in the SRAM-based lookup pipeline by using a technique known as write bubbles[11], which essentially encode and encapsulate the updates into write bubbles to be performed at each stage of the pipeline. Nevertheless, only a single port of the SRAM modules is used for read and write in building lookup pipelines in the past[11-12]. This means that write bubbles may lead to disruption to the IP lookup process. Much work[11-12] has targeted the reduction of the number of write bubbles resulted from route updates. Fortunately, state-of-the-art FPGAs now integrate dual port SRAMs, capable of concurrent reading and writing (with the possibility to do a write immediately after a read has been completed without any collision). This can be exploited to solve the problem of disruption caused by updates.

In a virtual router context, several router instances, and thus multiple FIBs, must be accommodated. This clearly exacerbates the memory requirement issues of hardware lookup solutions[13]. Recent researches [14-15] have concentrated on techniques to merge different virtual routers FIBs into a single, "compressed" trie structure, with a view to reduce the total memory requirement of the lookup engine. Nevertheless, route updates in the current Internet are known to occur frequently, with peak update rates affecting thousands of prefixes per second [16]. In the presence of virtual routers, a same network event could trigger simultaneous updates to multiple FIBs, thus increasing the rate of updates to the hardware lookup engine. Unfortunately, merging several FIBs together usually results in complex data structures whose update mechanisms become very challenging.

In this paper, we propose a different view to the problem of hardware IP lookup engine design. Rather than using only one type of hardware solution: TCAM or SRAM-based, we mix these two in order to benefit from the positive points of each architecture without being hindered by their weaknesses. Our aim is to design a very fast lookup architecture that enables fast updates concomitantly. The core idea of our solution is to exploit an empirically observed structure in 1-bit tries built from real FIBs (see TABLE I for more details). This observed structure is as follows:

1) About 90% of all prefixes are stored in trie leaves[17], and are thus disjoint from each other.

2) When the leaf nodes are removed from the original trie, non-prefix internal nodes that only lead to those leaf nodes can also be removed, and we are left with a much smaller trimmed trie, which contains, on average, only about 12% of the nodes of the original trie.

The large disjoint prefix set, resulting from property 1 above, makes a TCAM the ideal component to look these up, as naturally disjoint prefixes do not impose any order constraints within the TCAM, making updates trivial (no entry movements are required and a single write access is sufficient for each update since no prefix duplication is introduced). The small trimmed trie resulting from the removal of the leaf prefixes from the original trie, which represents the set of prefixes that overlap with the above mentioned disjoint prefix set, need much less memory space and can be stored in an on-chip SRAM-based lookup pipeline in FPGA. We will refer to this small trimmed trie as "*the overlapping trie*". In fact, several such trimmed tries can easily be accommodated in SRAMs of an existing FPGA. Updating this SRAM-based pipeline is also trivial, by exploiting the dual port capabilities of SRAMs mentioned earlier.

TABLE I. ANALYSIS OF REAL ROUTING TABLES

FIB	# prefixes	# nodes of the trie	# leaf prefixes	# nodes of the trimmed trie
rrc00	368057	905941	332409 (90.31%)	110109 (12.15%)
rrc01	358925	880946	325667 (90.73%)	103326 (11.73%)
rrc03	355603	873608	322419 (90.67%)	102984 (11.80%)
rrc04	366656	903163	332962 (90.81%)	104169 (11.53%)
rrc05	358355	879902	324594 (90.58%)	104457 (11.87%)
rrc06	351919	863114	319654 (90.83%)	100819 (11.68%)
rrc07	361881	888468	327781 (90.58%)	106517 (11.99%)
rrc10	355106	871466	321995 (90.68%)	102833 (11.80%)
rrc11	361708	888394	327742 (90.61%)	105552 (11.88%)
rrc12	363761	895781	329377 (90.55%)	106584 (11.90%)
rrc13	363057	894876	328942 (90.60%)	106024 (11.85%)
rrc14	361232	885979	327160 (90.57%)	105475 (11.90%)
rrc15	359326	880902	325154 (90.49%)	104536 (11.87%)
rrc16	366711	903062	331674 (90.45%)	108509 (12.02%)

In this paper, we mainly target fast FIB updates in high-speed routers. For this purpose, we propose a hybrid lookup architecture, composed of a TCAM-based lookup engine and an SRAM-based pipeline operating in parallel. The TCAM contains the disjoint prefixes and the SRAM-based pipeline contains the overlapping tries. We show that this hybrid approach results in fast lookup combined with easy and fast updates. We also show how our approach can be applied in the context of virtual routers, by simply prefixing IP addresses with a virtual router ID (VID), and performing the lookup on those "extended addresses".

We implement the proposed hybrid architecture on our PEARL hardware platform[18], and achieve a maximum throughput of 250 Million Lookups Per Second (MLPS). Comparative results show that the update overhead is significantly lower than that of previous work. Moreover, our TCAM memory can easily be dimensioned to achieve memory space utilization close to 100%.

The rest of the paper is organized as follows. In section II, we introduce our hybrid architecture and describe the optimizations for fast updates. In section III, we extend our approaches to support virtual routers. In section IV, we describe the architecture implementation on our PEARL platform and compare its performance with other techniques. We discuss some extensions in section V and conclude the paper in section VI.

II. ARCHITECTURE

In this section, we will describe our hybrid IP lookup architecture with fast updates in the context of a single router. We use 1-bit tries to illustrate the concepts. First, a 1-bit trie will be built from the FIB of the router, and a trie partitioning scheme will be applied to partition the trie into a large disjoint leaf prefix set and a small trimmed overlapping trie (we will use the terms *an overlapping trie* and *an overlapping prefix set* interchangeably through the paper). The large disjoint leaf prefix set is mapped into an external TCAM-based IP lookup engine, while the small trimmed overlapping trie is mapped into an on-chip SRAM-based IP lookup pipeline in FPGA.

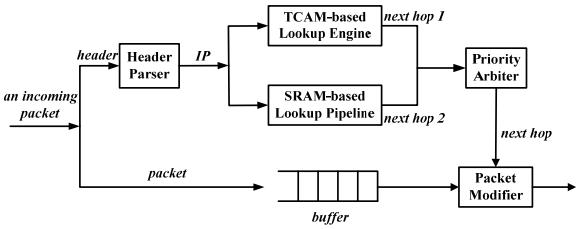


Figure 2. The hybrid IP lookup architecture

A. Trie Partitioning Scheme

Based on the observation described in the Introduction, an efficient trie partitioning scheme similar to the set-bounded leaf-pushing algorithm in [17] is applied to partition the 1-bit trie into two prefix sets. All the leaf prefixes in the trie are collected to form a large disjoint prefix set, and all the leaf nodes are removed from the trie. Then, we can further trim the remaining trie by removing non-prefix leaf nodes recursively until all the leaf nodes in the final trimmed trie are prefix nodes. Note that leaf pushing is not used in the trimmed trie in order to enable fast updates, which is the key difference between our trie partitioning scheme and that applied in [17].

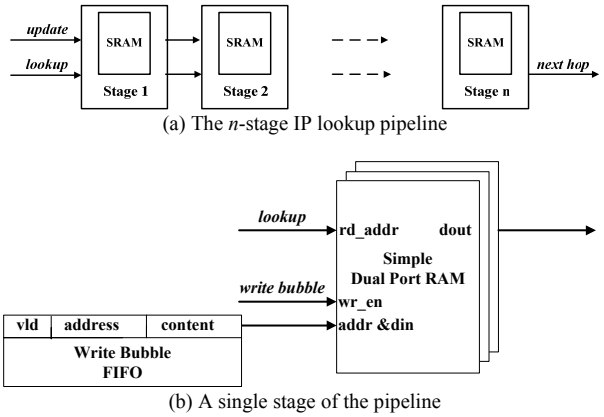
Fig. 1 illustrates the trie partitioning scheme. A 1-bit trie built from a sample FIB is shown in Fig. 1(a). In the trie, prefix P2, P4, P5, and P6 are leaf prefixes. All these leaf prefixes are moved to a disjoint prefix set (see Fig. 1(b)), and the leaf nodes 4, 7, 8, and 9 are deleted from the trie. Then the remaining trie can be further trimmed. For example, node 5 becomes a leaf node but it doesn't contain any prefix so it can be removed. The final trimmed trie is shown in Fig. 1(c) and represents the small overlapping prefix set (a small overlapping trie).

B. Overall Architecture

The hybrid IP lookup architecture is depicted in Fig. 2. It's composed of two IP lookup engines operating in parallel. The large disjoint leaf prefix set (*e.g.*, see Fig. 1(b)) is stored in the TCAM-based lookup engine, while the small overlapping trie (*e.g.*, see Fig. 1(c)) is mapped into the on-chip SRAM-based pipeline. The destination IP address of an incoming packet is extracted in the header parser module and sent to the two lookup engines to search in parallel. Meanwhile, the packet is stored in a buffer waiting for the next hop information. Since the length of the prefix matched in the disjoint prefix set is by design longer than that in the overlapping prefix set, the search result of the TCAM-based lookup engine has a higher priority than that of the SRAM-based lookup pipeline. Note, however, that a match does not necessarily exist in either lookup engine. After lookups in both lookup engines are completed, the priority arbiter module resolves the priority and determines the final next hop information. Thereafter the packet is read from the buffer and modifications are conducted based on the next hop information. Finally, the packet becomes ready to be scheduled to the corresponding output interface.

C. Optimizations for Fast Updates

Efforts are made in both lookup engines to optimize the update process.To achieve fast updates, only the large disjoint



(a) The *n*-stage IP lookup pipeline

(b) A single stage of the pipeline

Figure 3. The SRAM-based IP lookup pipeline

prefix set is stored in the TCAM-based IP lookup engine. In such a disjoint prefix set, a given IP address can only be matched by at most one prefix. This means that the prefixes can be stored in the TCAM without any order constraints. Therefore, the prefixes can be directly inserted in and deleted from the TCAM, and route updates do not need any entry movement. Moreover, the leaf prefix set is naturally disjoint and no prefix is duplicated. Hence, a single write access is enough for any route update in the worst case.

As explained in the Introduction, in the first generation of SRAM-based pipelines[11-12], write bubbles may lead to disruption to the IP lookup process since a write operation and a read operation could not be performed simultaneously on the same port of an SRAM. In this paper, we use new generation of FPGAs, like Xilinx FPGAs, that have dual port on-chip SRAMs[19]. These SRAMs can be configured into a simple dual port (SDP) mode where the SRAM has separate read and write ports. In this mode read and write can be performed simultaneously without any collision. Using this mode we design a pipeline with separate lookup and update paths in order to totally eliminate the disruption (see Fig. 3(a)). In this pipeline lookups are performed by only accessing the read port of the SRAM in each stage, while write bubbles are performed by only accessing the write port. In this way, IP lookups and write bubbles can be performed simultaneously in separate paths without collision.

Before a write bubble is injected into the pipeline, the data to be written to each stage of the pipeline are previously stored into a write bubble FIFO relative to each stage (see Fig. 3(b)). When the write bubble enters into the pipeline, it visits each stage for one clock cycle, and goes to the next stage. When a write bubble visits a stage, the data stored in the associated write bubble FIFO are written into the corresponding address when the valid flag is true. Through this scheme, the write bubble doesn't need to wait for the data and it can update each stage in just one clock cycle (that means it can go through the pipeline at the same speed as the lookup).

As a write bubble and an IP lookup can run at the same speed, and one write bubble is sufficient for a worst-case route update when using the 1-bit trie-based data structure for pipelining[12], an IP lookup never traverses the trie in an inconsistent state. More precisely, even when a lookup and a write bubble access the same node of the same stage simultaneously, the lookup still reads the old node before modification (thanks to the READ_FIRST feature of the SDP SRAM in Xilinx FPGA[19]), and this read-write order is kept when they both move to the next stage. Therefore, an IP lookup always accesses the trie in a consistent state during updating.

In summary, in our proposed architecture a single write access is sufficient for a worst-case route update in the TCAM-based lookup engine, and route updates have zero impact on the lookup process in the SRAM-based lookup pipeline.

D. Fast Incremental Updating Algorithms

We need to describe how an incremental route update is translated into updates in the TCAM-based lookup engine and the SRAM-based lookup pipeline. A route update can be classified into three main categories [17]: (1) insertion of a new prefix, (2) deletion of an existing prefix, and (3) modification of an existing prefix. The third type of the route update can easily be performed since it doesn't change the shape of the trie. However, the first two types are more complex. Insertion of a new prefix or deletion of an existing prefix may lead to prefix changes in both the disjoint prefix set and the overlapping trie.

To deal with this, we maintain an auxiliary 1-bit trie built from the FIB in the control plane of the router. The auxiliary 1-bit trie keeps track of prefixes stored in our hybrid architecture. An update operation consists of two phases. In the first phase, the route update is performed on the auxiliary trie and changes in the disjoint prefix set and the overlapping trie are found. In the second phase, optimized write accesses are applied to the hybrid architecture. In order to illustrate the incremental update process in our hybrid architecture, two complex update scenarios are shown in Fig. 4 and Fig. 5, respectively.

Fig. 4 illustrates the insertion of a new leaf prefix P7 (000*). After the insertion, prefix P2 turns into a non-leaf prefix and a new leaf prefix P7 appears. This results in three changes in the corresponding disjoint prefix set and the overlapping trie: (1) prefix P2 should be inserted into the overlapping trie, (2) prefix P2 should be deleted from the disjoint prefix set, and (3) prefix P7 should be inserted into the disjoint prefix set. After these changes are found in the control plane, P2 will be inserted into the SRAM-based pipeline, and then P7 will be inserted into the TCAM at the location where P2 was previously stored. The TCAM location where the leaf prefix is stored is recorded in the node data structure of the auxiliary 1-bit trie.

Fig. 5 illustrates the deletion of an existing leaf prefix P2 (00*). After the deletion, prefix P1 turns into a new leaf prefix.

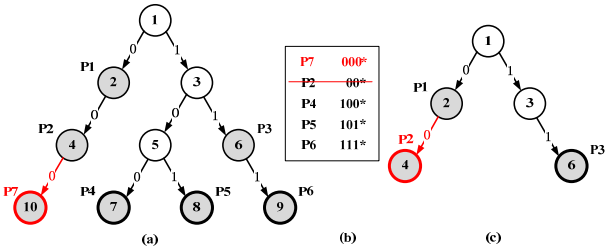


Figure 4. (a) Insertion of a new prefix, (b) its corresponding disjoint prefix set, and (c) its corresponding overlapping trie

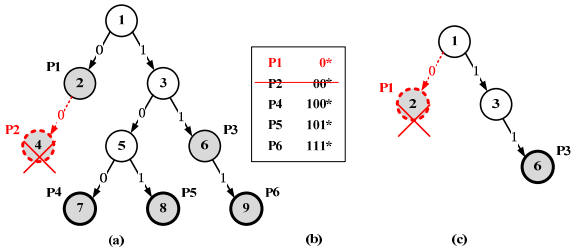


Figure 5. (a) Deletion of an existing prefix, (b) its corresponding disjoint prefix set, and (c) its corresponding overlapping trie

Input: Trie T, and Prefix P which is to be inserted to T.
Output: Changes in the disjoint prefix set S1 and the overlapping prefix set S2.

```
1   Insert prefix P into trie T, the new trie is T'
2   Find the longest prefix of P in T' : Prefix Q
3   if P is a non-leaf prefix in T'
4     Add P into S2;
5   else if P is a leaf prefix in T'
6     if Q is a non-leaf prefix in T
7       Add P into S1;
8     else if Q is a leaf prefix in T
9       Add Q into S2;
10      Del Q from S1, and add P into S1;
11    end if
12  end if
```

Figure 6. Algorithm: Insertion of a prefix

It leads to three changes in the corresponding disjoint prefix set and the overlapping trie: (1) prefix P2 should be deleted from the disjoint prefix set, (2) prefix P1 should be inserted into the disjoint prefix set, and (3) prefix P1 should be deleted from the overlapping trie. After detecting these changes in the control plane, P1 will be inserted into the TCAM at the location where P2 was previously stored, and thereafter P1 will be deleted from the SRAM-based pipeline.

Due to space limitation in this paper, we only illustrate two complex update scenarios but all scenarios are as easy to update. The complete insertion and deletion algorithms are presented in Fig. 6 and Fig. 7, respectively. Both of these algorithms are performed in software with a time complexity *O(l),* where *l* is the length of prefix *P* to be updated. In both algorithms, one route update generates at most one write operation to each lookup engine, and the order between the two write operations (if they exist) should be kept to avoid incorrect longest prefix matching during updating. For example, deleting prefix Q and inserting prefix P in the TCAM (see line 10 in Fig. 6) can be combined into one write operation by just overwriting prefix Q with P. Additionally, the execution of line 9 and 10 should be kept in the order shown in Fig. 6. Otherwise, prefix Q will disappear in both lookup engines temporarily, which may lead to incorrect longest prefix matching during updating.

III. LOOKUP FOR VIRTUAL ROUTERS

We described in previous section the hybrid IP lookup architecture for a single router. Nonetheless, the lookup architecture can naturally be extended to support virtual routers.

Input: Trie T, and Prefix P which is to be deleted from T.
Output: Changes in the disjoint prefix set S1 and the overlapping prefix set S2.

```
1   Delete prefix P from trie T, the new trimmed trie is T'
2   Find the longest prefix of P in T: Prefix Q
3   if P is a non-leaf prefix in T
4     Del P from S2;
5   else if P is a leaf prefix in T
6     if Q is a non-leaf prefix in T'
7       Del P from S1;
8     else if Q is a leaf prefix in T'
9       Del P from S1, and add Q into S1;
10      Del Q from S2;
11    end if
12  end if
```

Figure 7. Algorithm: Deletion of a prefix

A virtual router platform contains multiple FIBs; each FIB has the same feature as the FIB of a traditional non-virtual router. Therefore, the trie partitioning scheme is still suitable for each individual FIB in the virtual router platform. When each FIB is partitioned separately, multiple large disjoint prefix sets and relatively small overlapping tries are generated. We can further merge these disjoint prefix sets into a single one, and merge the overlapping tries into a single trie.

Several approaches have been proposed for merging prefix sets for virtual routers, *e.g.*, common prefix set[14] and virtual prefix technique [17]. For our purpose we have chosen the virtual prefix technique since it is simple and has a fast execution time[17]. In this scheme, by appending a unique virtual router ID (VID) before the prefix we get a virtual prefix. This ensures that the virtual prefix sets of all virtual routers are not overlapping. Hence, we can directly merge the virtual prefix sets of all virtual routers together to form a large prefix set. As an example, let's assign a VID 0 to the FIB shown in Fig. 1(a) and a VID 1 to the FIB shown in Fig. 8(a). Their corresponding prefix sets can be merged into two new prefix sets (see Fig. 9(a) and Fig. 9(b)).

Using the VID, all FIBs of virtual routers can be merged into a large disjoint prefix set and a relatively small overlapping trie (*e.g.*, see Fig. 9(a) and 9(b)). These two sets have the same feature as that in a single router. Therefore, the merged disjoint prefix set can be mapped into the external TCAM-based IP lookup engine, and the merged overlapping trie can be mapped into the on-chip SRAM-based IP lookup

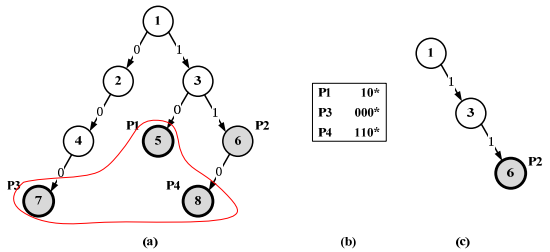


Figure 8. (a) Another sample trie, (b) the corresponding disjoint prefix set, and (c) the corresponding overlapping prefix set.

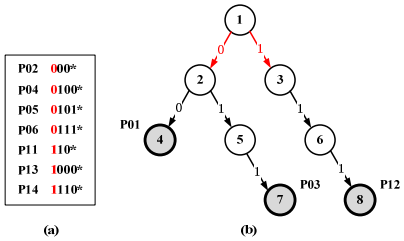


Figure 9. (a) The merged disjoint prefix set, and (b) the merged overlapping prefix set.

pipeline. This makes the architecture depicted in Fig. 2 suitable for virtual routers with a slight modification. The IP address used to search both lookup engines should be changed to a virtual IP address (VIP) by appending a VID to an IP address. This is performed in the header parser module shown in Fig. 2.

From this point, the update process in virtual routers becomes similar to that in a single router. When a route update is to be performed on one FIB of virtual routers, the same fast incremental updating algorithm described before is applied on the corresponding 1-bit trie to detect the changes in its disjoint prefix set and overlapping trie, with the difference that now the new prefix to be updated must be constructed by concatenating the prefix with the VID. Taking the insertion in Fig. 4 as an example, and assuming that they are relative to a virtual router instance with a VID 0, the changes in the final merged sets are as follows: (1) virtual prefix P02 (000*) should be inserted into the overlapping trie, (2) virtual prefix P02 (000*) should be deleted from the disjoint prefix set, and (3) virtual prefix P07 (0000*) should be inserted into the disjoint prefix set.

As mentioned before, one route update causes at most one write operation on each lookup engine for a single router. This remains valid for virtual routers; any route update in an FIB of virtual routers need at most one write operation on each lookup engine.

IV. Performance Evaluation

A. Analysis of Real Routing Tables

Fourteen real IPv4 routing tables have been collected from RIPE RIS Project[20] on 05/20/2011. Analysis is performed on these real routing tables to validate the advantage of the trie partitioning scheme. The analysis results are shown in TABLE I.

The number of prefixes and leaf prefixes in each FIB are shown respectively in column # *prefixes* and # *leaf prefixes*. We can see that for all the fourteen FIBs, more than 90% of the prefixes are leaf prefixes. This is expected since most of the prefixes are around 24-bit long, and most of them are disjoint leaf prefixes. The number of nodes in the original trie is represented in column # *nodes of the trie.* We applied the partitioning scheme. After moving the leaf prefixes into a disjoint leaf prefix set and trimming the trie further, the number of nodes remaining in the final trimmed trie is shown in column # *nodes of the trimmed trie*. The results show that after trimming, the number of remaining nodes is about 12% of that

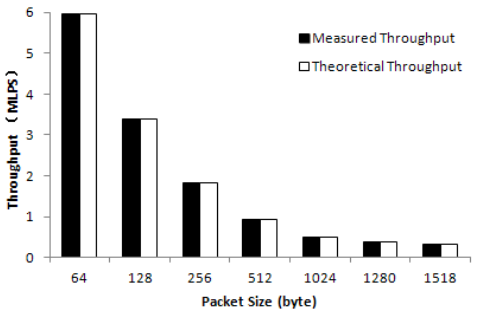


Figure 10. The throughput of the hybrid architecture

of the original trie. These observations confirm the initial empirical finding which is the base of the trie partitioning.

Based on the above analysis, the following conclusions can be drawn.

1) Using the partitioning scheme, most of the prefixes are moved to external TCAMs. Meanwhile, all of them are naturally disjoint and they can be stored without any order constraints. This feature can be used to guarantee fast updates in a TCAM.

2) After removing the leaf nodes, the amount of memory needed in the SRAM-based pipeline is reduced significantly. Hence, the memory size issue of on-chip SRAM-based pipelines in FPGA can be well addressed.

The above conclusions still hold for virtual routers as each router will have an FIB that will validate the above properties.

B. Throughput

We have implemented the hybrid architecture on our PEARL [18] hardware platform, which is equipped with a Xilinx Virtex-5 XC5VLX110T-1 FPGA and an IDT IDT75K72100 TCAM. The post place and route result in the FPGA shows a maximum clock frequency of 297 MHz (*i.e.*, 297 MLPS in the SRAM-based lookup pipeline). Besides, the TCAM has a theoretical maximum throughput of 250 MLPS. Hence, the implementation enables a maximum throughput of 250 MLPS, which exceeds largely the throughput requirement of 100G Ethernet. However, the PEARL platform we used has only four Gigabit Ethernet interfaces that need a maximum lookup rate of about 5.95 MLPS. We show in Fig. 10, the measured and theoretical throughput obtained over the PEARL platform with the proposed hybrid IP lookup architecture.

It is noteworthy, that it's hard to make a fair comparison with throughput measured in previous work, since the device types and optimization parameters of implementation tools are very different. However, the throughput of our implementation is clearly adequate for practical virtual routers.

C. Update Overhead

The number of TCAM write accesses per update is used as the metric to estimate the update overhead of TCAM-based engines. For the SRAM-based pipeline we use the number of disrupted lookup cycles per write bubble as the metric of comparison. We have chosen PLO_OPT/CAO_OPT[3], MIPS[4] and write bubbles in [11-12] as the comparison basis.

TABLE II. THEORETICAL COMPARISON OF THE NUMBER OF TCAM WRITE ACCESSES PER UPDATE

TCAM-based Engines	Maximum	Minimum
PLO_OPT	W/2 (16)	1
CAO_OPT	D/2 (16)	1
MIPS	2^{W-1} (2^{31})	0
Our Architecture	1	0

TABLE III. EMPIRICAL COMPARISION OF THE NUMBER OF TCAM WRITE ACCESSES PER UPDATE ON RCC00 ROUTING TABLE

TCAM-based Engines	Maximum	Average	Minimum
PLO_OPT	16	6.42	1
CAO_OPT	4	1.55	1
MIPS	247	1.15	0
Our Architecture	1	0.91	0

Theoretical comparison. In the best case, only one TCAM write access is required for each route update in both PLO_OPT and CAO_OPT[3], and zero TCAM write access is required for each update in both MIPS[4] and our architecture. However, the results in the worst case are quite different. In PLO_OPT[3], the prefix-length order should be kept and the empty space is arranged in the center of a TCAM. Therefore, a route update requires at most W/2 write accesses to the TCAM, where W is the maximum length of the prefixes (32 for IPv4). In CAO_OPT[3], the chain-ancestor order should be kept and the empty space is still arranged in the center. Therefore, a route update requires at most D/2 write accesses to the TCAM, where D is the maximum length of the chain. Theoretically, D may be up to W. MIPS[4] utilizes leaf pushing to convert the prefix set into an independent (disjoint) prefix set. However, leaf pushing may duplicate a prefix many times. In the theoretical worst case, a prefix could be duplicated to 2^{W-1} prefixes. Therefore, the maximum number of TCAM accesses for one route update is 2^{W-1}. In our hybrid architecture, the prefix set stored in the TCAM is naturally disjoint and prefix duplication is not required. One route update leads to at most one write access to the TCAM in any case. The theoretical comparison between different schemes is summarized in TABLE II.

Empirical comparison. We get from the RIPE RIS project [20] one of the publicly available routing tables rrc00 (see TABLE I) and one-hour update traces on it. The update traces contain 165,721 updates. Fig. 11 shows the running average of the number of TCAM accesses per update required for all the four compared TCAM update schemes as a function of the number of updates. The average in our proposed hybrid architecture remains persistently under one TCAM access (about 0.91) per update. This is expected since only one TCAM access is required for a leaf prefix update and zero TCAM access for a non-leaf prefix update. It can be seen that the average number of TCAM accesses in the hybrid scheme is much lower than that of all other three competing solutions. More importantly, the maximum number of TCAM accesses per update that directly affects the size of the packet buffer required in a lookup engine to avoid packet drops during updating, is precisely equal to one and significantly lower than that of competitor schemes (see TABLE III).

Obviously, the number of TCAM accesses per update in our proposed architecture can be proved to be optimal as at most a single write access per update to the TCAM is mandatory. This means that we can guarantee a minimum worst-case update overhead in the TCAM-based lookup engine. The summary of comparison results on routing table rrc00 is shown in TABLE III. Last but not least, even if MIPS is able to achieve a performance relative to an average number of write accesses per update that is close to 1, the worst-case overhead for a single update is very high (see TABLE II and TABLE III).

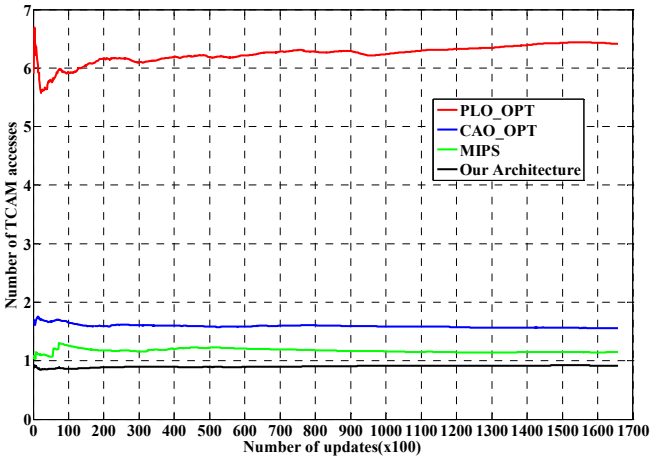


Figure 11. The running average of the number of TCAM accesses per update on rrc00 routing table.

In [11-12], write bubbles are used for route updates in SRAM-based pipelines. Each write bubble may disrupt the IP lookup process for one cycle in the worst case and minimizing the number of write bubbles reduces the update overhead. In our approach, we have addressed this challenge by devising a pipeline with separate lookup and update paths in order to totally eliminate the disruption to IP lookup process caused by write bubbles.

In summary, each route update leads to at most one write access in the TCAM-based IP lookup engine and has zero impact on the SRAM-based pipeline. Therefore, The update overhead is significantly lower than that of previous work[3-4, 11-12].

D. Memory Utilization

As explained before if the entire routing table was going to be managed by an SRAM-based pipeline, external memories would have been mandatory to support large routing tables in practice. However, due to the limited number of available I/O pins in FPGA, only a few external memories can be equipped. Hence, the utilization ratio of external memories becomes very important.

When external SRAMs are used for trie-based pipelines, a few large levels are moved into external SRAMs[7]. However, the size of those levels is variable and controlling the memory distribution among these stages is hard to achieve[7]. Therefore, the external SRAMs should be over-provisioned and memory waste can rarely be avoided. In the 2-3 tree-based routers[17], the last few stages of the SRAM-based pipeline are moved to external SRAMs. In these routers, a balanced tree named 2-3 tree is built so that the size of needed memory in level $i+1$ is about twice of that in level i. However, it is impractical to find

in the market external SRAMs with exact needed sizes. Due to this fact, it is hard to avoid memory waste when using 2-3 tree-based routers and the memory utilization ratio is usually low.

In our proposed architecture, the disjoint prefix set can be stored in external TCAMs without any order constraints. As a result, a disjoint prefix set can be mapped into a TCAM until it becomes full. Multiple external TCAMs can be cascaded to store more prefixes and each of those TCAMs can achieve a memory utilization ratio of 100% except the last TCAM. We should reserve enough empty space in the last one for further updating. Therefore, memory waste can be avoided. Additionally, the memory utilization among on-chip SRAMs in FPGA can be well balanced using the scheme proposed in [10].

V. DISCUSSIONS

A. Dual Pipelines

The SRAMs in Xilinx FPGA[19] are dual port. A read or write operation can be performed on each port alternatively. In this paper, one port of the SRAM is dedicated to read operations (lookups) and the other port is dedicated to write operations (updates). In such a pipeline with separate lookup and update paths, IP lookups and route updates can run simultaneously without any collision. Therefore, route updates have zero impact on lookups. However, by using each port of the SRAM both for lookups and updates, an alternative architecture of true dual pipelines can be built[17]. In such dual pipelines even though a route update may disrupt the lookup process in the same pipeline, it has no impact on the lookup process in the other pipeline. Therefore, the final forwarding performance of dual pipelines is much higher than that of a pipeline with separate lookup and update paths. However, we still choose the pipeline with separate lookup and update paths in our hybrid architecture for the following two reasons.

First, the forwarding performance of the pipeline with separate lookup and update paths is sufficient in our hybrid architecture. Generally, the clock frequency of an SRAM is higher than that of a TCAM. Given a typical SRAM with a clock frequency of 400 MHz and a typical TCAM with a clock frequency of 200 MHz, an SRAM-based pipeline can achieve a maximum throughput of 400 MLPS and a TCAM-based engine can achieve up to 200 MLPS. Obviously, in a hybrid architecture composed of a TCAM-based lookup engine and an SRAM-based pipeline operating in parallel, the final lookup performance is determined by the TCAM-based engine and it's not necessary to use dual SRAM-based pipelines.

Second, the implementation of dual pipelines is more complicated than that of the pipeline with two separate paths. In dual pipelines, each pipeline should be switched for lookups or updates. However, in the pipeline with two separate paths, the lookups and the updates run separately. Obviously, the structure of the two-path pipeline is simpler.

B. Memory Footprint

Although external TCAMs can be fully utilized and only 90% of the prefixes of the FIBs are stored in TCAMs, achieving a smaller memory footprint in a TCAM is desirable. For example, an existing large TCAM can accommodate up to 1024K 40-bit entries[21]. However, there are about 300K leaf prefixes in a single FIB (see TABLE I), which means that only leaf prefixes of about three virtual router FIBs can be accommodated in this TCAM. Therefore, the leaf prefixes stored in the TCAM should be compacted to support more FIBs in the context of virtual routers.

The compactions can be performed in two ways. First, leaf prefixes within a single FIB can be compacted. For example, if two leaf prefixes have the same parent node in a trie, and they have the same next hop, they can be replaced by their parent prefix. Second, leaf prefixes of different FIBs can be compacted. For example, if a prefix 110* with a VID 0 and a prefix 110* with a VID 1 coexist in the TCAM, and they have the same next hop, they can be merged to a single entry *110*. Indeed this issue exists for all TCAM management techniques.

On the other hand, memory balancing[8-10] and compact data structure like trie merging[14-15] can also be applied to the memory of on-chip SRAMs in FPGA to achieve a small memory footprint.

However, it's noteworthy that there is a trade-off between memory footprint and update overhead, since in the extreme, a very compact data structure may drastically increases the update overhead. This trade-off should be considered during compacting in practice.

As mentioned before, external SRAMs can also be used to extend the total memory size of on-chip SRAM-based pipelines. Each external SRAM should be over-provisioned and memory waste couldn't be avoided. However, an SRAM usually has a higher density than a TCAM. We are planning to study how to use external memories in an efficient way to support more FIBs after compaction in the near future.

C. Multi-bit Trie

In this paper, we have used a 1-bit trie structure. However, a multi-bit trie can be used to represent the final small overlapping trie before mapping it into the on-chip SRAM-based pipeline.

However, this brings new problems. When building a multi-bit trie, prefix expansion is needed in order to transform a prefix set into an equivalent one with allowed prefix lengths. However, prefix expansion may lead to node duplication [22]. Therefore, a single route update may need more than one write access on a single pipeline stage and multiple write bubbles may be required for a complete route update. In order to avoid incorrect longest prefix matching, no IP lookups are allowed to be injected into the pipeline until all the write bubbles belonging to a single route update are completed. Hence, route updates may lead to disruption to the IP lookup process and zero impact on the lookup process can no longer be guaranteed. This explains why we don't use the multi-bit trie to represent the overlapping prefix set.

VI. CONCLUSIONS

In this paper, we mainly focus on the FIB update challenge for high-speed routers. An efficient trie partitioning scheme is applied to convert a 1-bit trie into a large disjoint leaf prefix set and a small overlapping trie. This partitioning is motivated by

the observation that more than 90% of prefixes in the 1-bit trie are naturally disjoint leaf prefixes and can be easily mapped into external TCAM-based lookup engine. Thus, entry movements can be totally avoided and no prefix is duplicated, which results in a single write access for each update of a leaf prefix. Additionally, the memory management of TCAMs can be significantly simplified since a prefix in a disjoint prefix set can be stored in a TCAM at any available location. Therefore, we do not need to reserve empty space in each TCAM at special locations, and thus achieve a utilization ratio of TCAMs close to 100%. In other words, we only need to reserve some empty space in the last TCAM for further updating, and the remaining TCAMs can be fully utilized.

After removing the leaf nodes, the remaining trie can be further trimmed resulting in an overlapping trie that contains only about 12% of the nodes of the original trie. The overlapping trie is thereafter implemented in an SRAM-based pipeline with significantly lower memory requirement. In the context of virtual routers, multiple such overlapping tries can be accommodated in the on-chip SRAMs of existing FPGAs. Moreover, by exploiting the dual port SRAMs in Xilinx FPGA, we design an SRAM-based pipeline with separate lookup and update paths that enable simultaneous lookup and update operations without any collision. Therefore, route updates have zero impact on our dual-path SRAM-based pipeline.

The fast incremental updating algorithms guarantee that, in any case, any route update in the original 1-bit trie leads to at most one write access in our TCAM-based lookup engine, and at most one write bubble in our SRAM-based lookup pipeline (we can ignore the update overhead in our SRAM-based lookup pipeline since updates have zero impact on lookups). Therefore, we only need to lock the TCAMs for the time of at most one write access during each update. This update overhead is significantly lower than that of previous work.

In the context of virtual routers, a virtual router ID is assigned to each FIB and a simple merging scheme is applied. Then, the hybrid architecture can be well scaled to support virtual routers. Meanwhile, the update overhead of each route update stays the same as that in a single router.

The performance evaluation shows that the throughput is sufficient for 100G Ethernet routers, the update overhead is significantly lower than that of previous work, and the utilization ratio of most external high-capacity memories can be up to 100%. While the memory consumption of our proposed scheme is reasonable, we will study, as future work, compact data structures that can be applied to improve memory efficiency in both engines, while retaining the fast update property of the architecture.

ACKNOWLEDGMENT

This work was supported by National Basic Research Program of China under grant No. 2012CB315801, the NSFC-ANR pFlower project under grant No. 61061130562, NSFC under grant No. 61133015 and No. 60903208, and the Instrument Developing Project of the Chinese Academy of Sciences under grant No. YZ200926.

REFERENCES

[1] W. Jiang, Q. Wang, and V. K. Prasanna, "Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup," in *INFOCOM'08: Proceedings of the 27th Conference on Computer Communications*, pp. 2458-2466, 2008.

[2] Z. J. Wang, H. Che, M. Kumar, and S. K. Das, "CoPTUA: Consistent policy table update algorithm for TCAM without locking," *Ieee Transactions on Computers,* vol. 53, pp. 1602-1614, Dec 2004.

[3] D. Shah and P. Gupta, "Fast updating algorithms for TCAMs," *IEEE Micro,* vol. 21, pp. 36-47, 2001.

[4] G. Wang and N. F. Tzeng, "TCAM-based forwarding engine with minimum independent prefix set (MIPS) for fast updating," in *ICC'06: 2006 IEEE International Conference on Communications, Vols 1-12*, pp. 103-109, 2006.

[5] V. Srinivasan and G. Varghese, "Fast address lookups using controlled prefix expansion," *Acm Transactions on Computer Systems,* vol. 17, pp. 1-40, Feb 1999.

[6] S. Sikka and G. Varghese, "Memory-efficient state lookups with fast updates," *Computer Communication Review,* vol. 30, pp. 335-347, Oct 2000.

[7] W. Jiang and V. K. Prasanna, "Towards practical architectures for SRAM-based pipelined lookup engines," in *INFOCOM IEEE Conference on Computer Communications Workshops*, pp. 1-5, 2010.

[8] F. Baboescu, D. M. Tullsen, G. Rosu, and S. Singh, "A tree based router search engine architecture with single port memories," in *ISCA'05: Proceedings of the 32nd International Symposium on Computer Architecture*, pp. 123-133, 2005.

[9] S. Kumar, M. Becchi, P. Crowley, and J. Turner, "CAMP: fast and efficient IP lookup architecture," in *ANCS'06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, San Jose, California, USA, pp. 51-60, 2006.

[10] W. Jiang and V. K. Prasanna, "A memory-balanced linear pipeline architecture for trie-based IP lookup," *15th Annual IEEE Symposium on High-Performance Interconnects, Proceedings,* pp. 83-90, 2007.

[11] A. Basu and G. Narlikar, "Fast incremental updates for pipelined forwarding engines," in *IEEE INFOCOM 2003*, pp. 64-74.

[12] J. Hasan and T. N. Vijaykumar, "Dynamic pipelining: Making IP-lookup truly scalable," in *Proc. ACM SIGCOMM 2005*, pp. 205-216.

[13] K. Huang, G. Xie, Y. Li, and A. X. Liu, "Offset addressing approach to memory-efficient IP address lookup," in *IEEE INFOCOM Mini-Conference*, pp. 306-310, 2011.

[14] J. Fu and J. Rexford, "Efficient IP-address lookup with a shared forwarding table for multiple virtual routers," in *CoNEXT'08: Proceedings of the 2008 ACM CoNEXT Conference*, Madrid, Spain, pp. 1-12, 2008.

[15] H. Y. Song, M. Kodialam, F. Hao, and T. V. Lakshman, "Building scalable virtual routers with trie braiding," in *INFOCOM'10: Proceedings of the 29th Conference on Computer Communications*, pp. 1-9, 2010.

[16] *The BGP Instability Report.* Available: http://bgpupdates.potaroo.net/instability/bgpupd.html

[17] H. Le, T. Ganegedara, and V. K. Prasanna, "Memory-efficient and scalable virtual routers using FPGA," in *FPGA'11: Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, Monterey, CA, USA, pp. 257-266, 2011.

[18] G. Xie*, et al.* PEARL: A programmable virtual router platform. *IEEE Comm. Magazine, Special Issue on Future Internet Architectures: Design and Deployment Perspectives, 2011.*

[19] *Xilinx FPGA*. Available: http://www.xilinx.com/

[20] *RIPE RIS Raw Data*. Available: http://www.ripe.net/data-tools/stats/ris/ris-raw-data.

[21] NetLogic, "NL9000 RA knowledge-based processors," 2009.

[22] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *Ieee Network,* vol. 15, pp. 8-23, Mar-Apr 2001.